Appendix G: PC Based COE (DRAFT)

This appendix describes support for a Windows 95® and a Windows NT® based COE. (Windows® 3.1 and Windows for Workgroups 3.11 are not supported.) The COE concept is not specific to Unix, or any other operating system or windowing environment. However, certain adjustments to COE implementation details are required to support differences between the PC environment (use of '\' versus '/' in naming directories, byte swapping, etc.) and Unix, as well as to take advantage of features offered in the PC environment (e.g., registry).

This appendix should be viewed as an extension of the information contained in the rest of the document. The compliance requirements, descriptor information, and tools all apply except as modified and extended by this appendix.

This appendix is in provided in draft form. It describes the present approach as the COE is extended to include PC platforms for GCCS and GCSS. It is intended to be used for planning and initial implementation by those developers who will be building a PC COE-based system. An update to this appendix will be provided as part of the PC COE release.

G.1 Universal Desktop

Many operational sites have a large investment in PCs and PC software, or are planning significant PC acquistions. Introduction of a Unix-based command and control system poses challenges for such sites in how to best reuse existing computing assets to minimize capital, training, maintenance, administration, and other related costs. The most difficult technical challenge is that some functions are effectively operating system specific because they may only exist for one operating system (e.g., access to the JOPES database), because of strong operator preference (e.g., Microsoft Word® versus Applix®), or because of the availability of alternatives from the commercial market (e.g., spreadsheets, briefing support).

One approach is to implement a homogeneous network, or to divide work between operators in such a way that any individual operator deals with only those functions supported by a single operating system. This approach is not practical because the "optimum" platform to use is frequently application specific. Imposing an organization or dictating work responsibilities to accommodate system limitations is likewise undesirable and impractical.

A second approach is to provide operators with both a Unix platform and a PC platform. This approach is often neither viable nor desirable due to cost considerations, or the lack of available desk space for two monitors and keyboards. Moreover, data sharing between the two systems is difficult or impossible for even simple operations such as cut and paste.

This problem is not unique to the GCCS and GCSS programs, or the COE concept. It has been recognized in commercial industry as well where operators and administrators must manage and use a heterogeneous network. The solution proposed by commercial industry is the concept of a *Universal Desktop*. A Universal Desktop is capable of utilizing resources of the entire network, regardless of vendor platform, while displaying and accessing the information from a single computer monitor, keyboard, and mouse. Advances in distributed client/server technology, and advances in windowing software, provide the foundation necessary to realize the Universal Desktop concept. The Universal Desktop concept is maturing, but there are not yet any commercially available products which have widespread acceptance. There are several distinct approaches to the Universal Desktop, each with its own set of advantages, limitations, and advocates.

In the COE context, an operator on a Unix platform needs to have access to native Unix applications as well as selected PC-based office automation applications. An operator on a PC platform needs to have access to native Windows applications, and a selected number of Unix-based applications.

Universal Desktop approaches applicable to the COE may thus be broadly grouped into three fundamental categories:

1. Remote execution of client applications on a server, but local display of results.

2. Local execution of client applications with instruction set emulation as necessary for selected applications.

3. Native execution of client applications through porting as necessary for selected applications.

The third approach is the COE method. Limited support is provided for the first two non-COE approaches, but their wide spread use is not recommended. COTS software and licenses are the responsibility of the site for any required software, regardless of the approach selected. Sites which elect to use the first two approaches will require a combination of the first two if access from both PC and Unix platforms is required.

Note: It is typically *client* applications that need to be directly accessible to an operator, not *server* applications.

G.1.1 Remote Execution

Remote execution is the ability to run an application on a platform different from the operator workstation. If the application creates any displayable results, the results are displayed locally on the operator workstation. If the remote platform uses the same operating system and windowing software as the local platform, the issues are relatively straightforward and are not discussed in detail in this appendix. When the remote platform and the local platform do *not* use the same operating system and windowing software, the technical challenges are more difficult.

Ideally, the results should be displayed using the same "look and feel" as the native windowing software. That is, if a PC workstation remotely executes a Unix client application, the results should be displayed in the Microsoft Windows "look and feel." Unfortunately, this is usually not possible because the client application is written for a specific windowing environment, while the APIs and communications protocols are significantly different between X Windows and Microsoft Windows.

The two subsections which follow address the problem from the perspective of which platform is the remote platform. In the first, applications are executed remotely on a Unix platform but displayed locally on a PC running Microsoft

Windows. In the second, applications are executed remotely on a PC running Windows NT while results are displayed locally on a Unix platform running X Windows. In both cases, applications on the local machine are accessible and will not be discussed in detail. The approaches outlined are driven by availability of commercial products and should be viewed as secondary choices to native execution.

G.1.1.1 Unix Remote Execution

COE-based Unix segments utilize X Windows and Motif for GUI support. X Windows provides a convenient mechanism that supports remote execution of Unix applications, but allows displaying results locally on a PC workstation. Figure G-1 shows the principles involved. The approach requires an X server resident on the PC (Hummingbird® eXceed®, AGE® Xoftware32®, etc.) which is launched as a Microsoft Windows application. From within the X server resident on the PC, a remote login session is established with a client application on the Unix server. The X Windows `DISPLAY` environment variable on the Unix server is set to point to the X server running on the PC. This causes X display requests from the application to be transparently routed, through the underlying X Windows protocol, to the X server on the PC.

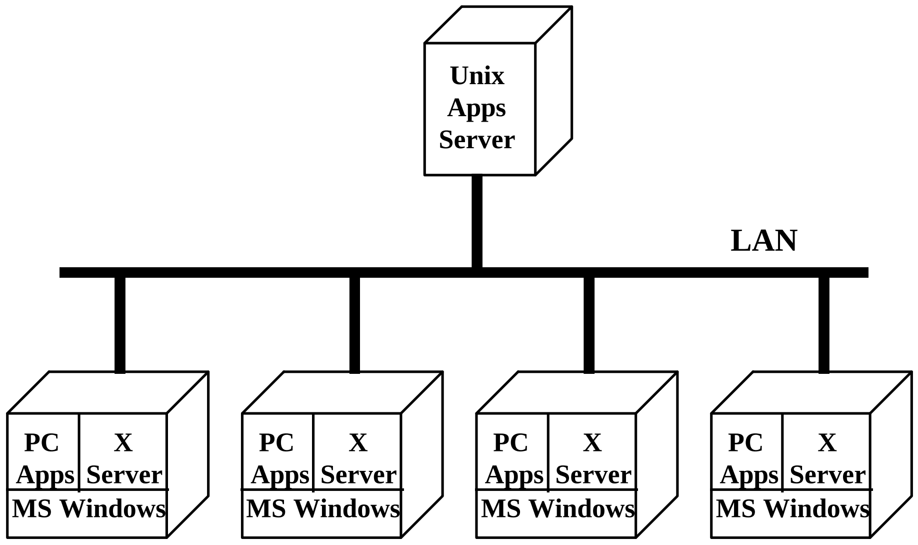

Figure G-1: Unix Remote Execution

The COE provides an account group, `RemoteX`, which contains the necessary files to establish an appropriate runtime environment on the Unix platform. The `DISPLAY` environment variable is automatically set and software is in place to limit the number of users who may access the Unix applications server at one time. User profiles and login accounts are administered from the Unix platform. The Unix desktop software presents the same interface to the PC based user as it does to the Unix based user, and provides the same level of auditing and security as for Unix based users.

Modifications to client applications are not normally required for this approach. However, this approach alone does not provide access to PC applications from a Unix workstation.

Hardware requirements for this approach are dependent upon which PC X server software is selected. Refer to specifications provided by the vendor. Procurement and installation of the appropriate PC X server and associated network software is the responsibility of the site.

DISA provides support for this approach, but does not recommend it due to:

¥ increased LAN loading,

¥ increased load on the Unix application server,

¥ the requirement for two GUI interfaces (X Windows and Microsoft Windows) on the PC,

¥ no access being provided to PC applications from the Unix platform, and

¥ limited data sharing between PC Windows and X Windows (e.g., cut and paste).

G.1.1.2 PC Remote Execution

Figure G-2 represents the approach taken by the commercial product WinDD®. WinDD is based on Windows NT 3.5 and was developed by a commercial vendor under a licensing arrangement with Microsoft. WinDD is completely compatible with NT, but extends NT to support multiple users and display of a complete Windows NT desktop within an X window.

With this approach, PC applications are loaded on one or more NT-based application servers, while Unix applications are loaded on one or more Unix based application servers. Some of the Unix applications may also be distributed or duplicated on sufficiently powerful Unix workstations. Unix workstations access Unix applications via X Windows, while PC applications are accessed through WinDD software. WinDD is loaded on both the NT applications server and individual workstations. On the workstations, NT applications appear within an X window controlled by WinDD software. The effect of this approach is the same as the preceding subsection, except that remote execution is on a PC instead of a Unix platform, and results are displayed locally on an X display instead of a PC. PC workstations may access PC applications either locally or from the server, but no access is provided to Unix applications from the PC workstations.

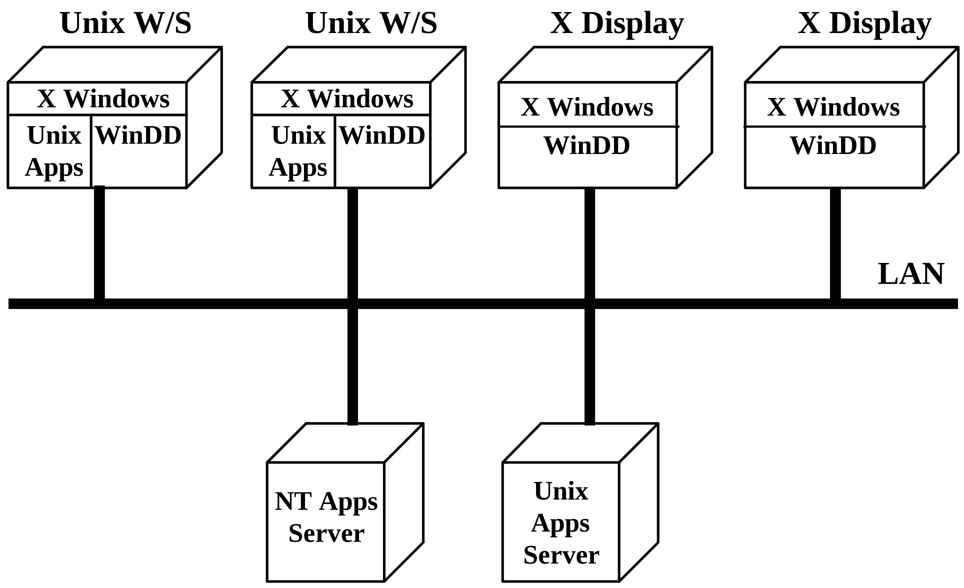

Figure G-2: PC Remote Execution

Sites may purchase WinDD from DISA for Unix platforms. DISA provides WinDD packaged as a COTS segment for simplified installation, and it is already configured to properly run on Unix workstations within the COE environment. The site must purchase and install WinDD for the NT server from the commercial vendor, and is responsible for WinDD software licenses. Application software running on the NT server is also the responsibility of the site.

Hardware requirements for WinDD are summarized below. Refer to vendor supplied WinDD specifications for a complete set of requirements.

Intel®-based System

(see *Microsoft Windows NT Hardware Compatibility List #4094* for minimum hardware requirements to run NT)

¥ 32MB RAM minimum. Add 4MB for each additional user, or 8MB for each additional "power user"

- 200MB minimum hard disk storage. Additional space for NT applications and user data are application dependent.

- VGA or SVGA graphics card compatible with Windows NT 3.5

WinDD Terminals

¥ 4-bit color only is supported

¥ 2 MB RAM per Windows NT desktop above memory requirements for other Unix applications

DISA provides support for this approach, but does not recommend it due to:

¥ increased LAN loading and decreased responsiveness, especially when NT applications are heavily mouse or graphics intensive (screen savers should be disabled),

¥ the cost of additional hardware requirements for PC systems running WinDD, and

¥ potential compatibility problems between applications available under Windows 3.1 and Windows NT.

G.1.2 Emulation

The SunSoft Wabi® product allows Windows 3.1 and Windows for Workgroups 3.11 applications to be loaded and executed directly on Unix platforms. The product translates, at runtime, calls to Microsoft Windows APIs to the equivalent X Windows APIs, and emulates 8x86 instructions. This approach provides an operator access to Unix and PC applications, but without the need to have a PC system. However, emulation on a Unix platform is slower than native execution on the PC, and the approach does not provide access to Unix applications from a PC.

Wabi does not currently support Windows 95 or Windows NT products. Sites are further cautioned that not all Windows 3.1/3.11 products are supported under Wabi. Products which require specialized device drivers (sound cards, video drivers, etc.) may not operate correctly. Products which require MS-DOS® either for installation or operation require purchase of a MS-DOS emulator which is not provided with Wabi.

Refer to vendor product specifications for the most current list of Wabi-compliant products.

Wabi requires the installation of either Windows 3.1 or Windows for Workgroups 3.11. Wabi assumes that software will be installed directly from floppies, but DISA provides a utility which allows software installed from floppies on one Unix platform to be transferred to another.

Sites may purchase Wabi and Windows directly from DISA. DISA provides Wabi and Windows pre-packaged as COTS segments for simplified installation from tape, and they are already configured to properly run on Unix workstations within the COE environment. The site is responsible for software licenses and any additional Windows application software.

Vendor specifications should be consulted for Wabi hardware requirements. The vendor advertised minimum disk space is on the order of 20-30 MBytes and 20-60 MBytes swap space. Minimum recommended memory is 32 MBytes.

DISA provides support for Wabi, but does not recommend it due to:

¥ a limited number of supported Windows applications,
¥ the sluggish performance of emulation approaches in general,
¥ the requirement to load Windows as well as Wabi,
¥ limited support for device drivers, and
¥ potential security holes in mixing Windows and Unix.

G.1.3 Native Execution

The COE approach to the Universal Desktop is to provide native execution of applications on each supported platform. Figure G-3 illustrates the approach. Client applications on the Unix platform use X Windows for GUI support, and access COE servers through a set of APIs. PC client applications use Microsoft Windows 95 or Windows NT for GUI support. PC applications access COE servers through a set of APIs on the PC which route requests across the LAN through Unix APIs to the appropriate server. Responses are passed back across the LAN and to the application as a response from the PC based APIs.

There are several advantages to this approach:

¥ Client applications use the same set of APIs whether executing on a Unix or a PC platform.

¥ Operators have a single desktop using the native GUI support (e.g., X Windows for Unix platforms, Windows for PC platforms).

¥ No special programming or site administration techniques are required.

¥ No additional software products must be purchased.

¥ Sites may allocate workstations by balancing availability, cost, preference, and operator workload as appropriate.

The remainder of this appendix describes the PC-based COE from the perspective of native execution. No specialized hardware or programming techniques are required. Applications to be accessible from the PCs are loaded directly on PC workstations. Sites will utilize Unix platforms as appropriate for database servers and as servers for critical COE components (message handling, comms, etc.). Sites may provide operators with Unix workstations or PC workstations as appropriate and as determined by operator work requirements.

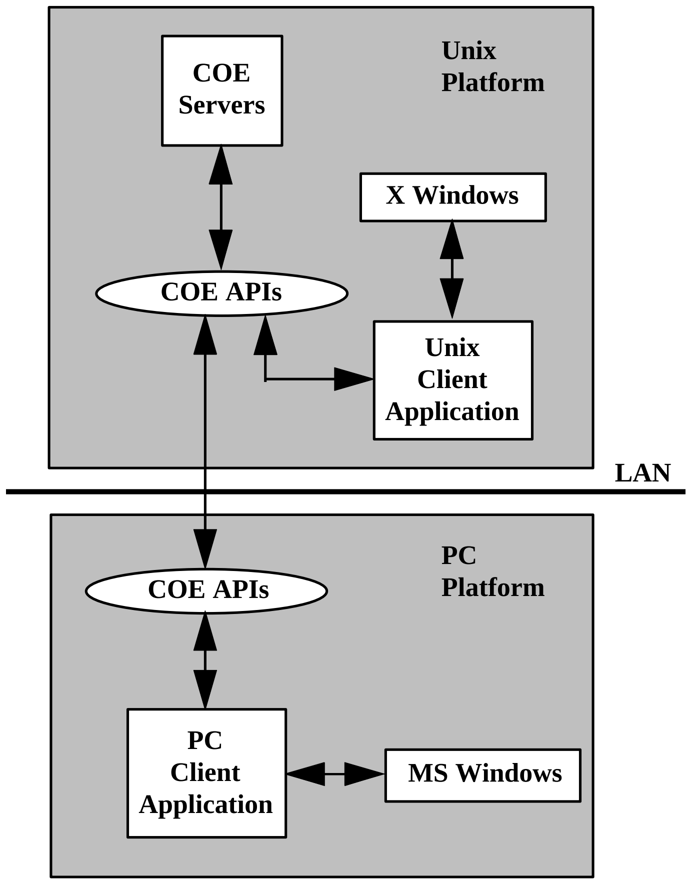

Figure G-3: Native Execution

G.2 Hardware/Software Requirements

The PC based COE requires Windows 95, or Windows NT version 3.51 or higher. Windows 3.1 and Windows 3.11 are not supported. All software development shall use Win32 APIs. Win16 APIs are not supported and shall not be used unless they are part of a COTS product for which there is no 32-bit alternative. Segments shall not duplicate functionality already provided by Windows.

All hardware shall be NT-compliant (as defined by the document *Microsoft Windows NT Hardware Compatibility List #4094*) whether Windows 95 or Windows NT is used. Minimum hardware requirements for a PC workstation are:

¥ 66 MHz 386 (66 MHz 486 recommended for Windows 95, 90 MHz Pentium recommended for Windows NT)

¥ 8 MBytes RAM for Windows 95 (16 MBytes recommended), 16 MBytes RAM for Windows NT (32 MBytes recommended)

¥ 200 MBytes disk space required (500 MBytes recommended)

¥ 3.5" floppy diskette drive

¥ LAN Interface card required to access Unix applications

¥ VGA or SVGA graphics card compatible with Windows NT, and capable of minimum 640x480 graphics in 256 colors

¥ 15" SVGA Monitor (17" recommended)

The following items are optional, but recommended. It is not necessary for every workstation to contain the additional hardware, but that a sufficient number of workstations on the LAN contain the additional hardware to meet site specific operational needs.

¥ 2x speed CD ROM (4x recommended)
¥ 16-bit Soundblaster® compatible card
¥ Tape drive for data archival
¥ HP Laserjet III® compatible laser printer
¥ Color printer for briefing slides

Note: Memory requirements stated here are the minimum for the kernel COE. 32 MBytes is the minimum for most mission applications. That is, for most mission applications not provided by commerical office automation products.

G.3 Disk Directory Structure

The PC-based COE uses the same basic directory structure shown in related figures from Chapter 5. However, Windows does not support symbolic links, and Intel-based computers store data bytes in a different order than other processors. This makes data sharing via disk files more difficult. This section describes the COE disk directory extensions required to support PCs.

Basic Directory Structure

The structure shown in Figure 5-1 is preserved for PCs. On disk drive `C`, subdirectory `\h` is created at the root level with subdirectories `COTS`, `AcctGrps`, `COE`, `data`, etc. The installation software will attempt to put segments on disk drive `C` first, but if it can not do so, it will load the segment on the next available hard disk. The installation software will create the directory `\h` and other subdirectories as required on additional hard disk drives.

The environment variable `INSTALL_DIR` is set to point to where the segment was loaded at install time, just as for Unix platforms, and includes the disk drive designation in the pathname. At runtime, the tool `COEFindDrive` (described below) can be used to find which disk drive contains the requested segment.

> **Note:** The installation software will *not* consider network disk drives in looking for the next available hard disk. Segments can only be installed on local hard disks, not floppies nor network drives.

Segment Directory Structure

A `Scripts` subdirectory is optional for PC segments because environment extension files are not supported, nor are they needed. Account group segments that need to establish global environment settings shall do so with a file called `GLOBALENV.BAT` contained in a `Scripts` subdirectory. The `GLOBALENV.BAT` file is used for the same purpose as the Unix `.cshrc` file. PC segments which need to establish local environment settings may do so through an `LOCALENV.BAT` file which shall be located in a `Scripts` subdirectory. Segments shall not include a `Scripts` subdirectory for any other purpose.

PC segments shall place all executables in the `bin` subdirectory. Segments which contain dynamic link libraries (DLL files) shall also place them in the `bin` subdirectory. With the exception of COTS segments, segments are not allowed to load DLL files in any other subdirectory.

PC segments which use private INI files shall store such files in the segment's `data\INI` subdirectory.

USERS Directory Structure

The PC COE uses the same operator directory structure as the Unix COE, as described in Chapter 5. Local operator accounts are specific to a single PC workstation, while global operator accounts are accessible from any PC on the network. However, operator accounts may not be mixed between Unix and PC platforms. Thus, an operator account, whether global or local, is either a PC operator account or a Unix operator account, but never both.

Global operator account subdirectories (e.g., `\h\USERS\global`) are physically located on a PC designated as the server. This directory is made accessible to other PCs on the network through the `share` command.

Environment variables `USER_HOME`, `USER_DATA`, and `USER_PROFILE` are used as described in Chapter 5. They are set by the appropriate account group.

Data Directory Structure

Chapter 5 defines data in terms of data scope. Local data is stored underneath `\h\data\local` while global data is stored underneath `\h\data\global`. Because data stored by the PC is not directly compatible with Unix platforms, an additional data subdirectory is created for storing PC *only* global data. This is the subdirectory `\h\data\PCglobal`. Segments shall follow the same rules for this directory as for the `\h\data\global` directory, except that only PC segments can access it. This subdirectory is physically located on a PC designated as the server and made accessible through the `share` command.

PCs may also access data stored in the `\h\data\global` subdirectory. However, this directory is *always* physically located on a Unix machine designated as a server. PC segments shall read and write data in the `\h\data\global` directory in network byte order. PC segments shall read and write data in the `\h\data\local` and `\h\data\PCglobal` directories in native PC byte order.

The installation software attempts to load data segments on the `C` disk drive. If there is insufficient room, it will load the data on the next available hard disk drive. PC segments must use the `COEFindData` tool, described below, to determine where data is actually stored.

Miscellaneous

1. Segments shall use file extensions that correspond to conventional Windows usage. That is, use `.EXE` for executables, `.DLL` for dynamic link libraries, `.TXT` for ASCII text files, etc.

2. Segments, excepting COTS segments, shall not set the Windows `path` environment variable.

3. Segments shall use the directory pointed to by the `TEMP` environment variable for temporary disk storage. This corresponds to using `/tmp` in Unix, and segments shall delete temporary files when an application terminates. All files in the `TEMP` directory are deleted when the computer is rebooted.

4. Segments shall not add a global "home" environment variable to the affected account group. Segments shall use `COEFindDrive` to determine the location of a segment, but may use a local "home" environment variable if desired (e.g., an environment variable defined and visible only within the segment) which is defined by results returned by `COEFindDrive`.

5. Environment extension files are not supported, nor required, in the PC-based COE.

6. `app-defaults` and `fonts` subdirectories are not meaningful in the PC COE. PC segments should not include these subdirectories. If they are included with a segment, the installation tools will not do any special processing for these subdirectories as is done for the Unix-based COE.

G.4 Account Groups

Account groups in the PC-based COE correspond to Windows Program Groups. The present PC COE does not include the `CharIF` or `DBAdm` account groups.

When the COE is loaded, the installation tools create program groups called `GCCS` (or `GCSS` for the Global Combat Support System), `SecAdm`, and `SysAdm`. The program items in each program group are determined as segments are loaded. Some program items, specifically for `SecAdm` and `SysAdm`, are provided by native Windows software and therefore will also be found in other program groups provided by Windows. This is done by creating duplicate icons which point to the same executable, not by creating multiple copies of the software.

As with the Unix COE, the specific icons and program groups available to an operator depend upon the operator profile.

G.5 Registry Usage

Microsoft Windows programs have traditionally created "INI" files to store configuration information. Windows 95 and Windows NT use a *registry* instead to store hardware parameters, configuration data, and Windows-maintained operator preferences. The registry is structured as a hierarchical database of keys organized into a tree structure. Each key can contain data items called *value entries* and can also contain additional *subkeys*. Subkeys are created through Windows APIs, and data values (string, binary, etc.) are associated with subkeys through APIs. In the registry, keys are analogous to directories while value entries are analogous to files.

The root level consists of several keys, the most important of which are:

`HKEY_LOCAL_MACHINE`	global settings for the system
`HKEY_CURRENT_USER`	current user's personal preferences
`HKEY_CLASSES_ROOT`	information for data file creation
`HKEY_CURRENT_CONFIG`	current machine configuration

Referencing a registry entry is similar to specifying a file's pathname. For example, the current version of Windows can be determined by the value of the registry key at

`HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion`.

Note that keys are case sensitive.

PC segments should use the Windows registry in place of INI files, except for storing operator preferences information. It is recommended that operator preferences be stored underneath `\h\USERS` to minimize porting problems between Unix and PC applications. Segments may use private INI files but if they are used, they shall be located in the segment's `data\INI` subdirectory.

Segments can not create root keys, but may create subkeys underneath the root keys as desired. In all cases, segments shall create subkeys using the convention "`SegType\SegDirName`" where *SegType* is either `COE`, `DATA`, or `SEGS` depending upon the segment type. `COE` is reserved for COE segments, `DATA` is reserved for data segments, and `SEGS` is for use by all other segments. *SegDirName* is the segment's directory name. Segments shall use the segment prefix to name all registry subkey entries.

For example, assume a software segment whose directory is SegA has a segment prefix SEGA. Assume the segments needs to store two pieces of information underneath HKEY_LOCAL_MACHINE\SOFTWARE:

1. the last coordinate system used (UTM, Lat/Lon, etc.) and
2. the last time a certain parameter was computed.

Then the required registry path is

```
HKEY_LOCAL_MACHINE\SOFTWARE\SEGS\SegA
```

and two appropriately named subkeys underneath this entry for storing value entries are SEGA_Last_Coord and SEGA_Last_Time.

Microsoft encourages use of the registry in some ways that are strictly forbidden in the COE because the COEInstaller tool performs some of these actions automatically. Refer to the subsection below on segment installation for actions performed by the installer. Segments, excepting COTS segments, shall not use the registry to duplicate any actions performed by the COE installation software:

¥ Segments shall *not* register "uninstall" information in the Uninstall key beneath CurrentVersion.

¥ Segments shall use the Processes descriptor to specify background processes. Segments shall *not* add values to either the Run or RunOnce keys beneath the CurrentVersion key.

G.6 Reserved Prefixes, Symbols, and Files

The segment prefixes listed as reserved in Chapter 5 are also reserved in the PC-based COE. In addition, the following segment prefixes are reserved:

`NT`	Windows NT segments
`WIN`	Generic Windows segments
`WIN95`	Windows 95 segments

The environment variables listed as reserved in Chapter 5 are also reserved in the PC-based COE. Segments may not create environment variables with the same name as any reserved environment variable. The following have no meaning in the PC-based COE, and are not guaranteed to be set:

```
DISPLAY
LD_LIBRARY_PATH
SHELL
TERM
TZ
XAPPLRESDIR
XENVIRONMENT
XFONTSDIR
```

All remaining environment variables listed in Chapter 5 are also defined for the PC-based COE.

The root level `AUTOEXEC.BAT` and `CONFIG.SYS` files are reserved files and shall not be modified by any segment, excepting COTS segments. Moreover, all windows INI files (specifically, `WIN.INI` and `SYSTEM.INI`) are reserved files and shall not be modified by any segment, excepting COTS segments. Segments should create and modify their own local INI files.

G.7 Programming Standards

Programming in the Windows environment is considerably different than the Unix/X Windows environment. This subsection details programming practices that are required to minimize problems in mixing the two environments.

G.7.1 File System

Windows NT supports four file systems: FAT, HPFS, NTFS, and CDFS. FAT (File Allocation Table) is the file system used by MS-MS-DOS, but it is extended in both Windows 95 and Windows NT (version 3.5 and later) to support long filenames. HPFS (High Performance File System) originated with OS/2®. NTFS (NT File System) originated with Windows NT as an improvement over both HPFS and FAT. CDFS (CD-ROM File System) is specific to CD ROM devices.

NTFS is the COE file system of choice because its security architecture corrects known problems in FAT. However, Windows 95 does not currently support NTFS, while the FAT file system is the only available file system for floppy disks. Therefore, the COE supports both FAT and NTFS. The type of file system in use should be transparent to most segments, but NTFS shall be used when there is a choice. Utilities are provided to convert files between FAT and NTFS.

A further complication is that NTFS filenames use the 16-bit *Unicode*® character set instead of 8-bit ASCII. Unicode is a technique for representing foreign alphabets (Japanese kanji, Chinese bopomofo, Greek, etc.). PC segments are not required to create Unicode strings, but segments must be able to read filenames which may be Unicode strings. This requirement is necessary because commercial products may be distributed on media which uses Unicode filenames, and because Windows NT uses Unicode strings internally. Windows 95 does not; it uses 8-bit ASCII strings internally.

Pathnames in Windows usually include a disk drive designation (e.g., `C:`). The disk drive containing the desired file may be located remotely on another machine. Windows allows symbolic names, called the *Universal Naming Convention* (UNC), to be given to remote paths so that an application need not know the workstation, disk drive, nor exact path to reach a particular file. UNC pathnames start with two backslashes (\\) followed by the server name, followed by the desired pathname and filename. Segments shall support the use of UNC pathnames.

To summarize,

1. Segments shall support the use of long filenames. Filenames are not allowed to contain embedded spaces, and should use file extensions as appropriate to conform to standard Windows usage.

2. Segments shall support use of UNC filenames.

3. Segments shall be Unicode aware, particularly with regards to filenames.

G.7.2 Dynamic Link Libraries

PC segments shall use *dynamic link libraries* (DLLs) to the maximum extent feasible. DLLs are located in the segment's `bin` subdirectory, except for COE segments. COE DLLs are located underneath the directory `\h\COE\bin` for all COE segments.

Windows originally exported DLL functions by assigning ordinal numbers to each exported function. Modules linked to DLL functions by ordinal number. However, later versions allowed linking to be by symbolic name rather than ordinal numbers. All PC segments shall link by symbolic name, and shall export DLL functions by symbolic name rather than ordinal numbers. The reason for this requirement is that ordinal numbers could change over time for exported functions, whereas the symbolic name will not.

G.7.3 Graphics

PC segments shall support VGA and SVGA resolutions, and shall use the Win32 API Graphics Display Interface (GDI) for creation of 2D graphics. This interface handles all calls made by applications for graphic operations and thus provides a standard interface for such calls. As a result, the Win32 GDI allows segments to be developed which are independent of the type of graphics output device in the end user's system. That is, segments need only make calls to standard graphic services provided by the Win32 subsystem regardless of the display, printer, or multi-media hardware used in the system.

To improve 2D graphics performance, the WinG library may be used. WinG is an optimized library designed to enable high-performance graphics techniques under Windows 3.x, Win32s, Windows NT, Windows 95, and future Windows releases. WinG allows the programmer to create a GDI-compatible HBITMAP with a Device Independent Bitmap (DIB) as the drawing surface. Programmers can use GDI or their own code to draw onto this bitmap, then use WinG to transfer it quickly to the screen. WinG also provides halftoning APIs that use the standard Microsoft halftone palette to support simulation of true color on palette devices. WinG is designed to easily facilitate extracting maximum graphics performance based on the runtime environment.

Segments shall use OpenGL APIs for 3D graphics. OpenGL is a software interface that allows the creation of high-quality 3D color images complete with shading, lighting, and other effects. OpenGL is an open standard designed to run on a variety of computers and a variety of operating systems. OpenGL consists of a library of API functions for performing 3D drawing and rendering. The core library contains low-level functions for graphics primitives, matrix transformations, lighting, shading, coloring, and texture mapping. A utility library is also available which supports commonly performed tasks such as drawing spheres and cylinders, building nonuniform rational B-spline (NURBS) curves and surfaces, and specifying 3D viewing volumes.

G.7.4 Fonts

Windows supports three different kinds of font technologies to display and print text: raster, vector, and TrueType®. The differences between these fonts reflect the way that the *glyph* for each character or symbol is stored in the respective font resource file. In raster fonts, a glyph is a bitmap that Windows uses to draw a single character or symbol in the font. In vector fonts, a glyph is a collection of line endpoints that define the line segments Windows uses to draw a character or symbol in the font. In TrueType fonts, a glyph is a collection of line and curve commands as well as a collection of hints. Windows uses the line and curve commands to define the outline of the bitmap for a character or symbol in the TrueType font. Windows uses the hints to adjust the length of the lines and shapes of the curves used to draw the character or symbol. These hints and the respective adjustments are based on the amount of scaling used to reduce or increase the size of the bitmap.

Raster and TrueType fonts are device independent, while vector fonts are not. TrueType fonts provide both relatively fast drawing speed and true device independence. By using the hints associated with a glyph, application software can scale the characters from a TrueType font up or down and still maintain their original shape. Segments shall use TrueType fonts to take advantage of the increased performance, flexibility, and WYSIWYG screen to printer characteristics. Custom application specific fonts shall be avoided in favor of using industry standard fonts wherever possible.

G.7.5 Printing

PC segments shall use the built in printing facilities provided by Windows. This includes using the Windows supplied printer common dialog box for configuring a printer, selecting print quality, selecting the number of copies, etc. All access to the printer shall be through Windows APIs.

Developers should be aware that some Win32 APIs are available only in Windows NT. Developers may use these APIs, but must ensure that the segment still operates correctly in a Windows 95 environment. As appropriate, PC segments should support drag and drop printing.

G.7.6 Network Considerations

UNC Filenames

PC segments shall support UNC filenames to access network shared drives and directories. If necessary, a segment can use the WinNet APIs to determine if a pathname is a network pathname.

The COE contains three pre-defined shared directories: `\h\data\PCglobal`, `\h\data\global`, and `\h\USERS\global`. The proper UNC filename to use for these three directories is determined by accessing registry subkeys underneath `HKEY_LOCAL_MACHINE\HARDWARE` as follows:

```
COE\Shared\data_PCglobal    \h\data\PCglobal
COE\Shared\data_global      \h\data\global
COE\Shared\USERS_global     \h\USERS\global
```

PC segments which create network sharable services or devices shall store UNC information in the registry. The subkey shall be either `COE\Shared` or `SEGS\Shared` depending upon segment type. The subkey shall be located underneath `HKEY_LOCAL_MACHINE\HARDWARE` for hardware devices (e.g., disk drives) or `HKEY_LOCAL_MACHINE\SOFTWARE` for software (e.g., servers). The segment shall document the proper registry information in the API documentation for the segment.

Network Byte Ordering

Computer architectures sometimes differ in the convention they use for how bytes are ordered in a word. This is the so-called "*big-endian, little-endian*" problem. Computers in which the *most* significant byte in a word is the leftmost byte use big-endian byte ordering. Computers in which the *least* significant byte in a word is the leftmost byte use little-endian byte ordering. Intel architectures use little endian byte ordering. When data is sent across the network, it is important to agree upon the same convention for byte ordering. The big-endian convention is also known as the *network byte order* and has been established as the industry standard.

The COE standard for byte ordering is network byte order. Segments shall ensure that all network data is transmitted in network byte order, except for certain data accessed on a network shared disk drive. Segments shall use APIs in the WinSock interface to ensure that data sent across the network is in network byte order. Segments shall store disk data accessible only by PCs in native PC byte order, but shall store disk data accessible by non-PCs in network byte order. The shared data directories and byte ordering are as follows:

`\h\data\PCglobal`	PC native byte order. Data here is shared, but is restricted to only PCs.
`\h\data\global`	Network byte order. Data in this directory may be accessible from a Unix platform as well as PCs.
`\h\USERS PC`	Native byte order. Data located here is specific to operator login accounts. Since a login account is either for Unix or a PC but never both, this data is platform specific.

Network Communications

Windows NT supports four transport layer protocols:

NetBEUI	provides compatibility with existing LAN Manager, LAN Server, and MS-Net installations.
TCP/IP	provides compatibility with standard Unix environments and a routable protocol for wide area networks.
Data Link Control (DLC)	provides an interface for access to mainframes and printers attached to networks.
AppleTalk®	provides interoperability with Macintosh networks.

TCP/IP is the COE standard network protocol. Segments shall perform network communications through WinSock APIs. Communications shall be designed to operate asynchronously to ensure that the server or application does not "hang" while waiting for a response.

G.7.7 Threads

PC segments should use threads to allow concurrent processing. Synchronization of threads may be achieved through critical sections, semaphores, mutexes, or events. Segments shall *not* use polling as a synchronization technique. Segment threads shall sleep at a synchronization point and allow the operating system to "wake up" the thread when the event requested occurs. This requirement prevents a thread from needlessly wasting CPU cycles and adversely impacting system performance.

G.7.8 Miscellaneous

The following statements apply to all new segment development. COTS segments may not meet all mandatory requirements, but shall be documented where they do not fulfill a mandatory requirement. To the extent possible, segments should conform to the requirements stipulated by Microsoft for allowing an application to use the Windows Logo.

Mandatory

1. Segments shall be "close aware." This means that the segment must enable the Close command and periodically check the close flag through the Query Close function.

2. Segments shall use common control and common dialog functions contained in `COMCTL32.DLL` and `COMDLG32.DLL`.

3. As appropriate, segments shall support cut and paste operations through the clipboard.

4. As appropriate, segments shall support drag and drop operations.

5. Segments shall support 16x16, 32x32, and 64x64 icons.

6. Segments shall *not* use MS-DOS functions.

7. Segments shall operate under both Windows NT and Windows 95. The segment shall degrade gracefully if it uses APIs found only in Windows 95 while running in a Windows NT environment, and vice versa.

8. Segments shall use *only* Win32 APIs.

9. Segments shall support long filenames and UNC.

10. Segments shall be Unicode aware.

Optional

1. Segments should run the Windows SDK tool `PORTTOOL.EXE` to identify potential problems with how Windows APIs are being used.

2. Segments should define the `STRICT` constant when compiling Windows code. This enables strict type checking during compilation.

3. Segments should avoid using environment variables. The registry or local INI files are preferred alternatives.

4. Developers are encouraged to use message crackers contained in `WINDOWSX.H`. Message crackers are a set of macros that makes code more readable, simplifies porting, and reduces the need to do type casting.

5. As appropriate, segments should register icons for document types and provide a viewer to allow the shell to display them. This is done through the `HKEY_CLASSES_ROOT` registry. Refer to Microsoft documentation for the required procedures.

G.8 Segment Installation

Segment installation follows the same sequence as for the Unix environment, and is defined in Chapter 5. As segments are installed on the PC, `COEInstaller` creates registry entries underneath `HKEY_LOCAL_MACHINE\SOFTWARE` corresponding to segment type. Assuming `SegDir` is the segment's directory name, the following registry keys are created:

`COE\SegDir`	for COE segments
`DATA\SegDir`	for data segments
`SEGS\SegDir`	for all other segments

These registry keys are deleted automatically when the segment is deleted.

`COEInstaller` sets the environment variables `INSTALL_DIR`, `MACHINE_CPU`, and `MACHINE_OS` for use in the `PreInstall.BAT` and `PostInstall.BAT` descriptors. The installer also stores the location where the segment was loaded in the subkey `SegDir\SegPath`. The value of this subkey includes the disk drive where the segment was loaded, but it can not be accessed until after segment loading is completed.

Segments shall not perform any operations (e.g., create registry entries, uninstall a segment) that are performed by the COE installation software.

G.9 PC COE Descriptors

The descriptor files defined in Chapter 5 apply to the PC-based COE as well. This section describes only differences between the Unix and PC environments.

PC segments are required to use `SegInfo` for descriptors; that is, PC segments may not use individual descriptor files since these are obsolete. All obsolete conventions are explicitly invalid for PC segments and are flagged as errors by `VerifySeg`.

When a home directory must be given, such as when specifying segment dependencies, a default disk drive may also be specified. If no default is specified, disk drive `C` is assumed. Pathnames must be given using '\' in conformance to the Windows environment.

When a disk drive designation is given, it and any associated pathname must be enclosed in double quotes. This is required so that the tools can distinguish between use of ':' as a field delimiter for descriptor lines, or as a separator between a disk drive name and a directory pathname.

For example, a `Requires` descriptor entry for a segment on drive `D` under the directory `\h\SegA` would be described as

```
segname:prefix:"D:\h\SegA":[version{:patch}]
```

AcctGroup

PC account groups must omit the *shell* parameter. It has no meaning in Windows.

COEServices

The `$GROUPS` and `$PASSWORDS` keywords are not supported for PCs. `VerifySeg` generates a warning if a segment descriptor contains these keywords.

DEINSTALL

To conform to Windows file extension conventions, `DEINSTALL` is renamed `DEINSTALL.BAT` for PCs.

FileAttribs

Because file permissions are different between the Unix and PC environments, `FileAttribs` is operating system specific. The COE tool `MakeAttribs`, when run on a PC, will create a proper `FileAttribs` file for PC segments. C style `#ifdef` preprocessor statements may be used to combine a Unix and PC `FileAttribs` descriptor.

Hardware

The *diskname* field for the `$PARTITION` keyword must be a disk drive name. For example, to indicate that a segment requires 20MB on the `F` disk drive, the proper `$PARTITION` statement is

```
$PARTITION:"F:":20480
```

Network

The `Network` descriptor is not supported for PCs. `VerifySeg` will issue a warning if a `Network` descriptor is found for a PC segment.

PostInstall

To conform to Windows file extension conventions, `PostInstall` is renamed `PostInstall.BAT` for PCs.

PreInstall

To conform to Windows file extension conventions, `PreInstall` is renamed `PreInstall.BAT` for PCs.

PreMakeInst

To conform to Windows file extension conventions, `PreMakeInst` is renamed `PreMakeInst.BAT` for PCs.

ReqrdScripts

Environment extension files are not supported for PCs. Therefore, the `ReqrdScripts` descriptor is not supported for PCs. `VerifySeg` will print a warning if this descriptor is present.

Requires

The pathname given for the `$HOME_DIR` keyword may include a disk drive designation. If a disk drive is not specified, the installation tools will load the segment in the directory indicated, but on the first available disk drive.

G.10 Executing Remote versus Local Segments

Remote execution of PC segments is not currently supported.

G.11 PC COE Tools

The COE Tools defined in Appendix C are also provided in the Windows environment, and operate in the same manner. This subsection describes differences in the tools between the Unix and PC environments.

COEFindData

`COEFindData` is a PC only tool. Its purpose is to determine where a data segment was loaded in case it could not be loaded on drive `C` because there was insufficient room. The syntax is the same as for `COEFindSeg`.

COEFindDrive

`COEFindDrive` is a PC only tool. The syntax is the same as for `COEFindSeg`. It returns the name of the disk drive, which may be a network drive, that contains the specified segment.

COEInstaller

The COE installation software is modified to allow segments to be loaded from floppy diskettes.

COEUpdateHome

Since environment extension files are not required in the PC COE, `COEUpdateHome` is not available for PCs.

MakeInstall

`MakeInstall` is modified to allow segments to be written to a floppy diskette. Compression/decompression is performed using the Microsoft File Compression Utility.

SegCatalog

This tool is not currently provided for the PC environment.

G.12 PC Segments and CSRS

There are no special requirements for submitting PC segments to CSRS. The tool `mkSubmitTar` is available on the PC, and it operates as described in Appendix C. `mkSubmitTar` does compression via APIs from the Windows Lempel-Ziv Expansion functions contained in the `LZEXPAND.DLL` library. `submit` will electronically transmit a packaged segment to CSRS.

G.13 PC COE Compliance Checklist

The items below are in addition to the compliance checklist in Appendix B.

G.13.1 Standards Compliance (Level 1)

Standards Compliance

T F N/A 1. Hardware components are Windows NT compliant.

Operating System

T F N/A 1. The operating system is Windows 95, or Windows NT version 3.51 or higher.

G.13.2 Network Compliance (Level 2)

Operating System

T F N/A 1. The segment accommodates FAT, CDFS, and NTFS files.

Network

T F N/A 1. The segments uses native PC byte order for data internal to the PC, but uses network byte order for data external to the PC.

T F N/A 2. The segment uses native PC byte order to access `/h/data/local` and `/h/data/PCglobal`.

T F N/A 3. The segment uses network byte order to access `/h/data/global`.

G.13.3 Workstation Compliance (Level 3)

Windowing Environment

T F N/A 1. Unless a COTS segment, the segment uses only Win32 APIs to access Windows routines.

COTS Products

T F N/A 1. If a 16-bit COTS segment, there is no 32-bit alternative product.

Runtime Environment

T F N/A 1. The segment executes correctly under Windows 95 and Windows NT.

Miscellaneous

T F N/A 1. The segment supports VGA and SVGA resolutions.

T F N/A 2. The segment supports 16x16, 32x32, and 64x64 icons.

G.13.4 Bootstrap Compliance (Level 4)

Standards Compliance

T F N/A 1. Unless a COTS segment, the segment does not modify the root level `AUTOEXEC.BAT` or `CONFIG.SYS` files.

T F N/A 2. Unless a COTS segment, the segment does not modify any Windows INI files.

Runtime Environment

T F N/A 1. The segment is able to handle Unicode filenames.

G.13.5 Minimal COE Compliance (Level 5)

Standards Compliance

T F N/A 1. Segment creates all its subkeys underneath `SegType\SegDirName` where *SegType* is either `COE`, `SEGS`, or `DATA`, and *SegDirName* is the segment's directory name.

T F N/A 2. All segment subkeys are named with the segment prefix.

T F N/A 3. The segment supports UNC filenames.

Segment Descriptors

T F	**N/A**	1.	Unless a COTS segment, the segment uses the `Processes` descriptor to create boot time processes. It does not set the `Run` or `RunOnce` keys underneath `CurrentVersion`.

Process Compliance

T F	**N/A**	1.	Unless a COTS segment, the segment does not register "uninstall" information in the registry (e.g., subkey `CurrentVersion\Uninstall`)

G.13.6 Intermediate COE Compliance (Level 6)

Standards Compliance

T F	**N/A**	1.	All INI files used are local to the segment and are stored in the segment's `data/INI` subdirectory.

Operating System

T F	**N/A**	1.	The segment supports long filenames.
T F	**N/A**	2.	The segment uses filename extensions in accordance with standard Windows usage (`TXT` for ASCII files, `DLL` for dynamic link libraries, etc.).

G.13.7 Interoperable Compliance (Level 7)

Network

T F	**N/A**	1.	The segment determines the location for shared data through the registry.
T F	**N/A**	2.	The segment stores information about shared resources in the location specified in this appendix.

Miscellaneous

T F	**N/A**	1.	The segment does not duplicate any Windows functions.

G.13.8 Full COE Compliance (Level 8)

Windowing Environment

T F N/A 1. The segment supports cut and paste through the clipboard.

T F N/A 2. The segment uses common control and dialog functions from `COMCTL32.DLL` and `COMDLG32.DLL`.

T F N/A 3. The segment is close aware.

Runtime Environment

T F N/A 1. Local environmental settings are established through an `LOCALENV.BAT` file in the segment's `Scripts` subdirectory.

Miscellaneous

T F N/A 1. The segment supports drag and drop.

T F N/A 2. The segment uses the Windows print dialog box for selecting printer configuration parameters.

G.13.9 Recommended Guidelines

The items listed here are not mandatory, but are strongly recommended to minimize porting problems and upgrading to subsequent COE releases.

T F N/A 1. The segment links to DLL functions by using symbolic names, not ordinal numbers.

T F N/A 2. The segment exports DLL functions by symbolic name, not ordinal numbers.